work simulation 90min (120min) 21Q

deadline requirement

	requirement	deadline
a	5	4
b	4	3
c	3	3

ab ac

dealine

1 ~ 5 most effective 5 1 least effective

amazon

1 video team

2 instant message

3

4 twenty one video log bug

book api

tell me more

tell me more

" "

api

API API due

due

book api deadline

digital

bug, bug

bug internal bug

complaints, Internal

test 20

It is not

helpful

Amazon recommendation system item

issue issue Germany

user name proxy name

amazon log

amazon log

bug bug in error log

case media network complaints,

invalid recommendation 404 / german

recommendation / invalid recom /

log username

log fi le book recommendation invalid,

.

deadline 2

4 demo

deadline

requirement deadline

requirement deadline

API Deadline

requirement

api api requirement

deadline cross team work team

customer feature deadline

bug log

.

3 1-2 4-5

15 Customer Complaints on Recommendation Service Priya Senior SDE

We have been receiving a lot of customer complaints over the past three days about problems on pages related to your new Recommendation feature. It is likely being caused by one of the three services you're using. There have been no recent updates to the services and we have ruled out all external factors. This issue is affecting customers, resulting in our product order rate dropping. We really need to solve this as quickly as possible. The reporting team generated an error report (see attached), let me know what you think. Thanks, Priya

Based on the data in the Error Rate over Time report, how would you respond? Please select only one response option.
Identify Service 1 as the problem Identify Service 2 as the
problem Identify Service 3 as the problem

I think Service 1 is the problem, but I would like to see another report to confirm ??

I think Service 2 is the problem, but I would like to see another report to confirm

I think Service 3 is the problem, but I would like to see another report to confirm?

I dont know which service is the problem. I want to request another report to get additional information.

16 New Product Design Hi

For the new social network integration product, Ravi is the senior engineer and Jane is the product manager. To help us decide what features to include in the time we have. I asked Ravi to estimate the time to develop each feature and I asked Jane to provide a rough prioritization of each feature. I put their responses in a chart on the Wiki. The file is named "Social Network Integration Design". I'm really swamped today so can you take a look at the chart and give me your recommendation for what features we should include? We have 8 weeks to get something up and running.

Rate the effectiveness of each of the following options
F, G
A, B, D
A, C, D, G
A, D, F
A, C, F
A, C, D, G, H

Feature	time to develop	bene fi ts
A,	1 to 2 weeks	M
B	4 to 6 weeks	H
C	2 to 4 weeks	L
D	2 to 4 weeks	M
E	5 to 9 weeks	H
F	3 to 5 weeks	H
G	2 to 4 weeks	M
H	1 to 2 weeks	L

FG	3-5 H 2-4 M	4
ABD	1-2 M 4-6 H 2-4 M	2
ACDG	1-2 M 2-4 L 2-4 M 2-4 M	6
ADF	1-2 M 2-4 M 3-5 H	1
ACF	1-2 M 2-4 L 3-5 H	3
ACDGH	1-2 M 2-4 L 2-4 M 2-4 M 1-2 L	5

17 Last part of code Aaron

Here's the last bit of code for the social network project. The code snippet is attached to this email. Sorry this took so long to finish, but it turned out to be more difficult than I thought. Can you just give it a quick check to make sure everything looks okay? I'm leaving for the day, so I'll see you at the demonstration in the morning! Thanks!

What is the problem with the Product.wasPurchasedByUser () method?
Please select only one response option.
It will run slowly because algorithm complexity is O (n ^ 2) It will run slowly on large datasets because more than one user could have purchased the same product.

It will run slowly because algorithm complexity is O (n ^ 3) It has performance issues.

What is the most effective way of improving the ShoppingCart () class for the long term? Please select only one response option.?
ShoppingCart should not override the user property every time new product is added to it.

Make ShoppingCart a property of Product class to improve performance. Change the design of ShoppingCart by removing ShoppingCart.user and making shopping cart a property of User instead.

Product should have getUser () method so that we know what user has added it to his / her shopping cart.

Please indicate whether the five tests within the ShoppingCartTest () class would pass or fail. Assume each unit test is independent of the others. Please provide a response for each test.

Test1 fail

Assert.assertTrue user1.getDefaultPaymentMethod () getIsDefault ().;
getDefaultPaymentMethod () null

Test2
pass
PaymentMethod method1 = new
PaymentMethod (PaymentMethodType.CREDIT_CARD, "myPersonalCard", true); Assert.assertSame method1,
user1.getDefaultPaymentMethod () Test3 pass

Assert.assertEquals Bob user2.getName () fail

Assert.assertEquals alice@foo.com user1.getEmail ()
email

Test4
pass
Assert.assertSame (thumbnail, myProduct.getImages (). Get (0)) Test5 fail

Assert.assertTrue (10.75 == product.getPrice ());
setPrice overload int double double
int
setPrice double integer getPrice integer

20 Problem with the website Jacob

Are you still here? I just learned there might be an issue with the website. This could be a real problem but I'm not sure what's going on. No one else is here right now so I could use your help. If you were to ask Jacob a question about the issue, how would you rate the importance of each of the following questions?

Do any other projects depend on fixing this problem? 4 How long will it take to solve this
problem? 5 How does this affect customers? 3 1

Are we receiving complaints from customers? 1 2 How many customers is this
affecting? 2
If I help you with this problem, will you help me finish my work today? 6

bug

bug bug

bug bug

coding 60min (70min)
we value fully working code more than partially working but efficient code

1 Round Robin
public static float waitingTimeRobin (int [] arrival, int [] run, int q) import java.util.LinkedList; import java.util.Queue; //
eg

// arrival_time = [0, 1, 4], execution_time = [5, 2, 3], q = 3 // average wait time = (7 - 5) + (5 - 3) + (10 - 7) / 3 =
2.3333333 // q is quantum

public class RoundRobin {
// SOL 1
public static float waitingTimeRobin1 (int [] arrival, int [] run, int q) { // corner case

if (arrival == null || run == null || arrival.length! =
run.length) {
return 0;} // use Queue to store Process type elements Queue <Process>
queue = new LinkedList <Process> (); int curTime = 0; int waitTime = 0;

// use a int type variable to track the next Process index int nextProIdx = 0;

while (! queue.isEmpty () || nextProIdx <arrival.length) {
if (! queue.isEmpty ()) {
Process cur = queue.poll (); // continue suming up the waitTime waitTime + =
curTime - cur.arriveTime; // based on Round Robin's principle, every round

time is limited within the certain quantum q
// if exceed time q, the not finished process should
be forced to be interrupted, and switch to the next process waiting in the Queue

curTime + = Math.min (cur.excuteTime, q); // if arrival time of the next process is
smaller
than current time
// means the next process should be pushed into
the Queue
for (int i = nextProIdx; i <arrival.length; i ++)
{
if (arrival [i] <= curTime) {
queue.offer (new Process (arrival [i],
run [i]));
nextProIdx = i + 1;} else {

break;}

```
                                        } // push the interrupted process into the tail of
the Queue
                                        if (cur.excuteTime> q) {
                                                queue.offer (new Process (curTime,
cur.excuteTime - q));}

                                } Else {
                                        // push element in arrival time array and
corresponding element in run time array into Queue
                                        queue.offer (new Process (arrival [nextProIdx],
run [nextProIdx]));
                                        // at the same update the current time point curTime = arrival [nextProIdx
                ++];}} return (float) waitTime / arrival.length;} // declare a Process type to store arrival time array
                and run time array




        public static class Process {
                int arriveTime; int
                excuteTime;
                Process (int arr, int exc) {
                        arriveTime = arr; excuteTime =
                exc;}}



        public static void main (String [] args) {
                int [] arrival1 = {0, 1, 4}; int [] run1 = {5, 2, 3}; int
                q1 = 3;
                int [] arrival2 = {0, 1, 3, 9}; int [] run2 = {2, 1, 7, 5}; int
                q2 = 2;


                System.out.print (waitingTimeRobin1 (arrival2, run2, q2));}}



2 Rotate Matrix
public class RotateMatrix {
        public static int [] [] rotate1 (int [] [] matrix, int flag) {
                if (matrix == null || matrix.length == 0 ||
matrix [0] .length == 0) {
                        return null;}


                int rows = matrix.length; int cols = matrix [0] .length; int [] [] right
                = new int [cols] [rows];
```

```
                int [] [] left = new int [cols] [rows]; if (flag == 1) {


                        for (int i = 0; i <rows; i ++) {
                                for (int j = 0; j <cols; j ++) {
                                        right [j] [rows - 1 - i] = matrix [i] [j];}} return right;} else if (flag == 0) {




                        for (int i = 0; i <rows; i ++) {
                                for (int j = 0; j <cols; j ++) {
                                        left [cols - 1 -j] [i] = matrix [i] [j];}} return left;} return null;}




        public static void printMatrix (int [] [] test) {
                for (int i = 0; i <test.length; i ++) {
                        for (int j = 0; j <test [i] .length; j ++) {
                                System.out.print ( "" + test [i] [j]);} System.out.println ();}}




        public static void main (String [] args) {
                int [] [] test = {{1, 2, 3, 4, 5},
                                {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15}};

                printMatrix (rotate1 (test, 0));}}
```

3 Find minimum path sum of a binary tree BST

BST

```
  10 / \ 5157
/ \ 310 / / \
1812
```

```
public class MinPathSum {
        // time O (n) space O (logn)
        public static int minPath4 (TreeNode root) {
                if (root == null) {
```

```
                return 0;}


            if (root.left == null && root.right == null) {
                return root.val;} int minSum =

            Integer.MAX_VALUE;
            Stack <TreeNode> pathNode = new Stack <TreeNode> (); Stack <Integer> pathSum
            = new Stack <Integer> (); pathNode.push (root); pathSum.push (root.val);! While
            (pathNode.isEmpty ()) {


                TreeNode curNode = pathNode.pop (); int curSum =
                pathSum.pop ();
                if (curNode.left == null && curNode.right == null) {
                    if (curSum <minSum) {
                        minSum = curSum;} else {

                        minSum = minSum;}} if

                (curNode.right! = null) {


                    pathNode.push (curNode.right); pathSum.push (curSum +

                curNode.right.val);} if (curNode.left! = null) {


                    pathNode.push (curNode.left); pathSum.push (curSum +

            curNode.left.val);}} return minSum;} //




    public static int minPath2 (TreeNode root) {
        if (root == null) {
            return 0;} if (root.left! = null && root.right == null) {


            return minPath2 (root.left) + root.val;} if (root.left == null && root.right! =

        null) {
            return minPath2 (root.right) + root.val;} return Math.min (minPath2 (root.left), minPath2

        (root.right))
+   root.val;
    } //

    public static int minPath1 (TreeNode root) {
        if (root == null) {
            return 0;} if (root.left == null && root.right == null) {


            return root.val;
```

```
} if (root.left == null) {

return root.val + minPath1 (root.right);} if (root.right == null) {


return root.val + minPath1 (root.left);} return root.val + Math.min

(minPath1 (root.left),
minPath1 (root.right));
}

public static class TreeNode {
int val; TreeNode left;
TreeNode right; TreeNode (int
x) {

val = x;}}}
```

4 insert value into a circle linked-list

int

sorted cycle linkedlist list

case insert

CNode start CNode

list duplicate CNode ascending order

dup test case 22/23 40min case

```
public static CNode insert7 (CNode myList, int n) {
// corner case if it is null if (myList == null) {

return new CNode (n);
} Else if (myList.next == myList) {// only one element
// add this node myList.next = new CNode (n);

// the next's next reference points back to the head
so forms a cycle myList.next.next = myList;

// check value and return the smaller one as head if (myList.val <n) {

return myList;} else {

return myList.next;
```

```
                }
        } Else if (n <myList.val) {
                // if it is the smallest element // find tail and append
                CNode cur = myList; while (cur.next = myList!) {


                        cur = cur.next;} // add value after the

                tail cur.next = new CNode (n);


                // set the appended node's next to original header cur.next.next = myList;

                // because this is smallest value! And that's another
reason we return List other than void
                return cur.next;} // find a position when node.val <n and node.next.val> n // or node.next ==
        head (largest) CNode cur = myList;



        while (cur.next! = myList && cur.next.val <= n) {
                cur = cur.next;} CNode curNext =

        cur.next; cur.next = new CNode (n);


        // made a copy of original next and assign it here cur.next.next = curNext;

        // return header position is unchanged return myList;}



    // SOL 2
    // case 1: pre.val ≤ x ≤ cur.val: // Insert between pre and cur.

    // case 2: x is the maximum or minimum value in the list:. // Insert before the head eg the head has the smallest
  value and its pre.val> head.val.

    // case 3: Traverses back to the starting point:. // Insert before the starting point public
     static CNode insert3 (CNode head, int x) {

        if (head == null) {
                head = new CNode (x); head.next =

            head; return head;} CNode cur = head;

            CNode pre = null; do { pre = cur;




                cur = cur.next; // case 1

                if (x <= cur.val && x> = pre.val) {
                        break;}
```

```
                        // case 2
                        if (pre.val> cur.val && (x <cur.val || x> pre.val))
{
                                break;}

                } While (! Cur = head); // when back to starting point, then
stop. For case 3

                CNode newNode = new CNode (x); newNode.next
              = cur; pre.next = newNode; return newNode;}



        public static class CNode {
                int val; CNode next;
                CNode (int x) {

                        val = x;}}



5 greatest common divisor
// Euclid algorithm //
                        m, n ∈ N                gcd (m, n) = gcd (n, m mod n)
        // SOL 1 recursion
        public static int getGCD1 (int [] arr) {
                if (arr == null || arr.length == 0) {
                        return 0;} if (arr.length <2) {


                        return arr [0];}


                int rst = arr [0];
                for (int i = 1; i <arr.length; i ++) {
                        if (arr [i]> 0 && rst> 0) {
                                rst = helper (rst, arr [i]);} else {

                                return 0;}} return

                rst;}



        private static int helper (int m, int n) {
                if (m% n == 0) {
                        return n;} else {

                        return helper (n, m% n);}} // time O (n),

             space O (1)
```

```
public static int getGCD3 (int [] nums) {
        if (nums == null || nums.length == 0 || nums [0] == 0) {
                return 0;}


        if (nums.length == 1) {
                return nums [0];} int rst =

        nums [0];
        for (int i = 1; i <nums.length; i ++) {
                if (nums [i] == 0) {
                        return 0;} // m is the divident //
                rst is the divisor m = [i] int nums; while
                (! M% rst = 0) {


                        int tmp = rst; rst = m% rst;
                m = tmp;}} return rst;}
```

6 Shortest Job First input

requestTimes [] request ; Durations [] request

requestTimes []

example1

requestTime: [0, 2, 4, 5]

duration: [7, 4, 1, 4] short

task fi rst p1 7 p3, 1 p3 p2 p5 average waiting time 4 test case .

example2:

requestTime: [0, 1, 3, 9]

duration: [2, 1, 7, 5]

average waiting time 0.5

// O (nlogn)

```
public static float SJFaverage2 (int [] request, int [] duration) {
        if (request == null || duration == null || request.length!
= Duration.length) {
                return 0;} int index

        = 0;
        int len = request.length; int waitTime = 0; int
        curTime = 0;

        PriorityQueue <process> heap = new
PriorityQueue <process> (new Comparator <process> () {
                public int compare (process p1, process p2) {
```

```
                                        if (p1.excTime == p2.excTime) {
                                                return p1.arrTime - p2.arrTime;} return p1.excTime -
                                p2.excTime;}});



                        while (! heap.isEmpty () || index <len) {
                                if (! heap.isEmpty ()) {
                                        process cur = heap.poll (); waitTime + = curTime -
                                        cur.arrTime; curTime + = cur.excTime;

                                        while (index <len && curTime> = request [index])
{
                                                heap.offer (new process (request [index],
duration [index ++]));}

                                } Else {
                                        heap.offer (new process (request [index],
duration [index]));
                                        curTime = request [index ++];}} return (float)
                        waitTime / len;}



// O (nlogn)
        public static float SJFaverage6 (int [] arrive, int [] execute) {
                if (arrive.length == 0) {
                        return 0;}


                PriorityQueue <process> heap = new
PriorityQueue <process> (arrive.length, new Comparator <process> () {
                        public int compare (process p1, process p2) {
                                if (p1.excTime == p2.excTime) {
                                        return p1.arrTime - p2.arrTime;} else {

                                        return p1.excTime - p2.excTime;}}});




                int curTime = 0; int waitTime =
                0; int i = 0;

                while (! heap.isEmpty () || i <arrive.length) {
                        if (heap.isEmpty ()) {
                                int at = arrive [i];
                                while (i <arrive.length && arrive [i] == at) {
                                        heap.offer (new process (arrive [i],
execute [i]));
                                        curTime = arrive [i]; i ++;}
```

```
                        } Else {
                                process p = heap.poll (); waitTime + = curTime -
                                p.arrTime; curTime + = p.excTime; for (; i <arrive.length;
                                i ++) {

                                        if (arrive [i] <= curTime) {
                                                heap.offer (new process (arrive [i],
execute [i]));
                                        } Else {
                                                break;}}}} return (float) waitTime
                        / arrive.length;}




        public static class process {
                int arrTime; int
                excTime;
                process (int x, int y) {
                        arrTime = x; excTime
              = y;}} // O (n ^ 2)



public static float SJFaverage3 (int [] request, int [] duration) {
                if (request == null || duration == null || request.length
== 0 || duration.length == 0) {
                        return 0;}


                int len = request.length; int [] end = new int
                [len]; int curTime = 0;

                for (int i = 0; i <len; i ++) {
                        if (i == 0) {
                                curTime = duration [0]; end [0] =
                            duration [0];} else {

                                int minDuration = Integer.MAX_VALUE; int curProcess = 0; for
                                (int j = 0; j <len; j ++) {

                                        if (end [j]! = 0) {
                                                continue;}


                                        if (request [j] <= curTime) {
                                                if (duration [j] <minDuration) {
                                                        minDuration = duration [j]; curProcess = j;}}
                                                else {
```

```
                                                    break;}}


                                        if (curProcess == 0) {
                                                curProcess = i;
                                                curTime = request [curProcess];}


                                                curTime = curTime + duration [curProcess]; end [curProcess] =
                                        curTime;}}


                        int waitSum = 0;
                        for (int i = 0; i <len; i ++) {
                                waitSum + = end [i] - request [i] - duration [i];}


                        return (float) waitSum / (float) len;}
```

7 count miss output
miss count
eg
size = 4, input array {1, 2, 3, 4, 5, 4, 1} 1 miss 2
miss 3 miss 4 miss

5 miss replace 1 4 hit
4
1 miss replace 2

```
        // SOL 3
         public static int count3 (int [] arr, int size) {
                if (arr == null || arr.length <1) {
                        return 0;}


                List <Integer> cache = new ArrayList <Integer> (); int cnt = 0;

                for (int i = 0; i <arr.length; i ++) {
                        if (cache.contains (arr [i])) {
                                int tmp = arr [i];
                                cache.remove (cache.indexOf (arr [i])); cache.add (tmp);} else {


                                cache.add (arr [i]); cnt ++;} if
                            (cache.size ()> size) {


                                cache.remove (0);
```

```
                    }} return
        cnt;}

// SOL 1?
 public static int count1 (int [] arr, int size) {
            if (arr == null) {
                    return 0;}


            List <Integer> cache = new ArrayList <Integer> (); int cnt = 0;

            for (int i = 0; i <arr.length; i ++) {
                    if (cache.contains (arr [i])) {
                            cache.remove (new Integer (arr [i]));} else {

                            cnt ++;
                            if (size == cache.size ()) {
                                    cache.remove (0);}} cache.add

                (arr [i]);} return cnt;} // SOL 2




 public static int count2 (int [] arr, int size) {
            if (arr == null || arr.length <1) {
                    return 0;} List <Integer> list = new ArrayList <Integer> (); int cnt = 0;



            for (int num: arr) {
                    int index = list.indexOf (num); if (index == -1) {

                            cnt ++; list.add (num); if (list.size ()> size) {


                                    list.remove (0);}} else {


                            list.remove (index); list.add
                (num);}} return cnt;}
```

8 day change (cell growth)

int [] dayChange (int [] cells, int days), cells 8

cell, 0

1(inactive active coding 01).

cells [0] cells [len-1] 0

eg

cells: [1, 0, 0, 0, 0, 1, 0, 0] days:

1

output: [0, 1, 0, 0, 1, 0, 1, 0]

// time O (n * days) space O (n)

```
public static int [] change5 (int [] arr, int days) {
        if (arr == null || arr.length <= 1 || days <= 0) {
                return arr;} int len =

        arr.length;
        // preNum represents previous day's list int [] preNum = new int [len];
        preNum = arr;

        for (int cnt = 0; cnt <days; cnt ++) {
                int [] curNum = new int [len]; curNum [0] = preNum [1];
                curNum [len - 1] = preNum [len - 2]; for (int i = 1; i <len - 1; i
                ++) {

                        curNum [i] = preNum [i - 1] ^ preNum [i + 1];} preNum = curNum;} return

        preNum;}



public static int [] change3 (int [] arr, int days) {
        if (days <= 0) {
                return arr;}


        int cnt = 0; while (cnt <days) {

                int [] tmp = new int [arr.length]; for (int i = 0; i <arr.length; i ++) {

                        int left; int right; if (i - 1 <0) {


                                left = 0;} else {

                                left = arr [i - 1];}


                        if (i + 1> arr.length - 1) {
                                right = 0;} else {

                                right = arr [i + 1];}


                        if (left == right) {
```

tmp [i] = 0;} else {

tmp [i] = 1;}}

arr = tmp; cnt ++;} return

arr;}

9 Find path in maze

2D array, m * n 0 1 00

cheese 9 return 1 0

// backtracking // 1 path 0

wall

public static int maze3 (int [] [] grid) {

int rows = grid.length; int cols = grid [0] .length; int [] [] visited =

new int [rows] [cols]; if (! dfsHelper (0, 0, grid, visited)) {

return 0;} return

1;}

public static boolean dfsHelper (int i, int j, int [] [] grid, int [] [] visited) {

if (i> = 0 && i <grid.length && j> = 0 && j <
grid [0] .length && grid [i] [j] == 9) {

visited [i] [j] = 1; return true;}

if (isSafe (grid, i, j) == true && visited [i] [j] == 0) {

visited [i] [j] = 1;

if (dfsHelper (i - 1, j, grid, visited)) {

return true;} if (dfsHelper (i, j + 1, grid, visited)) {

return true;} if (dfsHelper (i + 1, j, grid, visited)) {

return true;} if (dfsHelper (i, j - 1, grid, visited)) {

return true;} visited [i]

[j] = 0; return false;}

```
                return false;}


        public static boolean isSafe (int [] [] grid, int i, int j) {
                if (i> = 0 && i <grid.length && j> = 0 && j <
grid [0] .length && grid [i] [j] == 1) {
                        return true;} return

            false;}

// 0 path 1 wall
        public static boolean isSafe2 (int [] [] grid, int i, int j) {
                if (i> = 0 && i <grid.length && j> = 0 && j <
grid [0] .length && grid [i] [j] == 0) {
                        return true;} return

            false;}


        public static void main (String [] args) {
                int [] [] grid = {
                                {9, 0, 1, 1, 1}, {1, 1, 0, 0, 1}, {0,
                                1, 0, 0, 1}, {0, 9, 1, 1, 1}};



                System.out.print (maze3 (grid));}}



// iteration bfs
        public static boolean maze2 (int [] [] grid) {
                if (grid == null) {
                        return false;} Queue <point> queue = new LinkedList <point> (); int [] [] mark =
                new int [grid.length] [grid [0] .length]; int [] dx = {0, -1, 1, 0}; int [] dy = {1, 0, 0, -1}; point
                start = new point (0, 0); queue.offer (start); while (! queue.isEmpty ()) {




                        point tmp = queue.poll (); mark [tmp.x] [tmp.y] = 1; if
                        (grid [tmp.x] [tmp.y] == 9) {

                                return true;
                        } Else if (grid [tmp.x] [tmp.y] == 0) {
                                continue;} else {

                                for (int i = 0; i <4; i ++) {
                                        if (tmp.x + dx [i]> = 0 && tmp.x + dx [i] <
grid.length && tmp.y + dy [i]> = 0 && tmp.y + dy [i] <grid [0] .length) {
                                                if (mark [tmp.x + dx [i]] [tmp.y + dy [i]]
== 0) {
```

```
                                                    queue.offer (new point (tmp.x +
dx [i], tmp.y + dy [i]));
                                                }}}}}
                        return false;}




        public static class point {
                int x; int y;

                point (int x, int y) {
                        this.x = x; this.y =
                y;}}



10 Check Subtree
Determine whether T2 is subtree of T1
// SOL 1
// time O (nm) or O (n + km) k is the number of occurrences of T2's root in T1

// space O (log n + log m) recursion
public static int check1 (TreeNode t1, TreeNode t2) {
                if (t2 == null) {
                        return 1;} if (subTree (t1, t2)) {


                         return 1;} else return

            -1;}

        public static boolean subTree (TreeNode r1, TreeNode r2) {
                if (r1 == null) {
                        return false;
                } Else if (r1.val == r2.val && matchTree (r1, r2)) {
                        return true;} return (subTree (r1.left, r2) || subTree (r1.right, r2));}




        public static boolean matchTree (TreeNode r1, TreeNode r2) {
                if (r2 == null && r1 == null) {
                        return true;
                } Else if (r1 == null || r2 == null) {
                        return false;
                } Else if (r1.val! = R2.val) {
                        return false;} else {

                        return (matchTree (r1.left, r2.left) &&
matchTree (r1.right, r2.right));
```

}}

public static class TreeNode {
int val; TreeNode left; TreeNode right;
public TreeNode (int x) {

val = x;}}

11 Valid Parenthesis

valid parenthesis string - 1 valid pair

import java.util.Stack; // time O (n), space O (n / 2) public static
int isValid (String s) {

if (s == null || s.length () == 0) {
return 0;}

Stack <Character> stack = new Stack <Character> (); for (int i = 0; i <s.length (); i
++) {
if (stack.isEmpty ()) {
stack.push (s.charAt (i));
} Else if (s.charAt (i) - stack.peek () == 1 ||
s.charAt (i) - stack.peek () == 2) {
stack.pop ();} else {

stack.push (s.charAt (i));}} if

(stack.isEmpty ()) {

return s.length () / 2;} else {

return -1;}}

public class Solution {
public boolean isValid (String s) {
if (s == null || s.length () == 0) {
return true;}

Stack <Character> stack = new Stack <Character> (); int len = s.length (); for (int i
= 0; i <len; i ++) {

char c = s.charAt (i); if (stack.isEmpty
()) {
if (c == ')' || c == ']' || c == '}') {
return false;}

```
                    stack.push (c); continue;}


                if (c == ')' && stack.peek () == '(' || c == ']' && stack.peek () == '['
                || c == '}' && stack.peek () == '{') {

                    stack.pop ();
                } Else if (c == '(' || c == '[' || c == '{') {
                    stack.push (c);} else {

                    return false;}} return
            stack.isEmpty ();}}




public class Solution {
    public boolean isValid (String s) {
        if (s == null || s.length ()% 2 == 1) {
            return false;}


        HashMap <Character, Character> map = new HashMap <Character, Character> ();

        map.put ( '(', ')'); map.put ( '[', ']');
        map.put ( '{', '}'); / *


        map.put ( 'a', '1'); map.put ( 'b',
        '2');
        * /
        Stack <Character> stack = new Stack <Character> (); for (int i = 0; i <s.length (); i
        ++) {
            char c = s.charAt (i); if (map.containsKey
            (c)) {
                stack.push (c);
            } Else if (map.containsValue (c)) {
                if (! stack.isEmpty () && map.get (stack.peek ()) == c) {
                    stack.pop ();} else {

                    return false;}}} return
        stack.isEmpty ();};}
```

12 reverse second half of linked list

mid mid mid + 1

```
public class ReverseList {
```

```
public static JNode reorder2 (JNode head) {
        if (head == null || head.next == null || head.next.next ==
null) {
                return head;}

        JNode fast = head.next; JNode slow =
        head;
        while (fast.next! = null && fast.next.next! = null) {
                fast = fast.next.next; slow = slow.next;} JNode
          pre = slow.next; JNode cur = pre.next; while (!
          Cur = null) {

                pre.next = cur.next; cur.next =
           slow.next; slow.next = cur; cur = pre.next;} return
           head;}

public static JNode reorder3 (JNode head) {
        if (head == null || head.next == null) {
                return head;} JNode pre = findMidPre
       (head); JNode right = pre.next; pre.next = null; right
       = reverse (right); pre.next = right; return head;}

public static JNode reverse (JNode node) {
        if (node == null) {
                return node;} JNode dummy = new
        JNode (0); while (! Node = null) {

                JNode tmp = node.next; node.next =
           dummy.next; dummy.next = node; node =
           tmp;} return dummy.next;}

public static JNode findMidPre (JNode node) {
        if (node == null) {
                return node;} JNode fast =

        node.next;
```

```
                JNode slow = node;
                while (fast! = null && fast.next! = null &&
fast.next.next! = null) {
                        fast = fast.next.next; slow = slow.next;} return

                slow;}



        public static class JNode {
                int val; JNode next;
                JNode (int x) {

                        val = x;}}



        public static void main (String [] args) {
                JNode n1 = new JNode (1); JNode n2 =
                new JNode (2); JNode n3 = new JNode
                (3); JNode n4 = new JNode (4); JNode n5
                = new JNode (5); JNode n6 = new JNode
                (6); n1.next = n2; n2.next = n3; n3.next =
                n4; n4.next = n5; n5.next = n6; n6.next =
                null; JNode x = reorder2 (n1); while ( x! =
                null) {




                        System.out.print (x.val); x = x.next;}}}
```

13 same tree

```
/ **
  *    Definition for binary tree
  *    public class TreeNode {
  *            int val;
  *            TreeNode left;
  *            TreeNode right;
  *            TreeNode (int x) {val = x;}
  *}
  * /
public class Solution {
        public boolean isSameTree1 (TreeNode p, TreeNode q) {
                if (p == null && q == null) {
                        return true;
```

```
        } Else if (p == null || q == null) {
                return false;
        } Else if (p.val == q.val) {
                if (isSameTree (p.left, q.left) && isSameTree (p.right,
q.right)) {
                        return true;} else {

                        return false;}} return

        false;}



    public boolean isSameTree2 (TreeNode p, TreeNode q) {
        if (p == null && q == null) {
                return true;} if (p == null || q == null) {


                return false;} Stack <TreeNode> s1 = new Stack <TreeNode> ();
        Stack <TreeNode> s2 = new Stack <TreeNode> (); TreeNode cur1 = p;
        TreeNode cur2 = q; while (true) {



                while (cur1! = null && cur2! = null) {
                        s1.push (cur1); s2.push (cur2); cur1 = cur1.left; cur2 =
                    cur2.left;} if (cur1! = null || cur2! = null) {



                        return false;} if (s1.isEmpty () && s2.isEmpty ()) {


                        break;} cur1 = s1.pop
                ( ); cur2 = s2.pop ();

                if (cur1.val! = cur2.val) {
                        return false;} cur1 =
            cur1.right; cur2 = cur2.right;} return
            true;}}
```

14 find out number of arithmetic sequence in array public class ArithmeticSeq {

```
    public static int find (int [] nums) {
            if (nums == null || nums.length <3) {
                    return 0;
```

```
        } int left = 0; int right = 1;


        int diff = nums [1] - nums [0]; int cnt = 0;

        while (right <nums.length - 1) {
            if (diff = nums [right + 1] -! nums [right]) {
                cnt + = (right - left - 1) * (right - left) / 2; if (cnt> 1000000000) {

                    return -1;} diff = nums [right + 1] - nums [right]; left =

            right;} right ++;} cnt + = (right - left - 1) * (right - left) / 2; return cnt>

            1000000000 -1: cnt;}?




    public static void main (String [] args) {
        int [] arr = {2,5,2,3,4,6,8,10,12,9,8,7,6,2,4,8}; System.out.println (find (arr)) ;}}
```

15 Amplitude

```
public class Amplitude {
    // 4
    public static int tmp = 0;

    public static int getAmplitude (TreeNode root) {
        if (root == null) {
            return 0;}


        int rst = dfs (root, root.val, root.val); return rst;}



    public static int dfs (TreeNode root, int min, int max) {
        if (root == null) {
            return 0;} min = Math.min (min, root.val);
        max = Math.max (max, root.val);


        if (root.left == null && root.right == null) {
            tmp = Math.max (tmp, max - min);} dfs (root.left, min,

        max); dfs (root.right, min, max); return tmp;} // bug 2
```

```
public static int getAmplitude2 (TreeNode root) {
        if (root == null)
                return 0;
        if (root.left == null && root.right == null)
                return root.val;
        Stack <TreeNode> pathNode = new Stack <TreeNode> (); Stack <int []> pathMinMax
        = new Stack <int []> (); pathNode.push (root);

        pathMinMax.push (new int [] {root.val, root.val}); int amplitude = Integer.MIN_VALUE;
        while (pathNode.empty ()!) {

                TreeNode crtNode = pathNode.pop ();
                if (crtNode.left == null && crtNode.right == null) {
                        int [] crtMinMax = pathMinMax.pop (); amplitude = amplitude>
                        (crtMinMax [1] -
crtMinMax [0]) amplitude:? (crtMinMax [1] - crtMinMax [0]);
                        continue;} if (crtNode.right! = null) {


                        int [] crtMinMax = new int [2]; crtMinMax [0] =
                        crtNode.right.val <
? PathMinMax.peek () [0] crtNode.right.val: pathMinMax.peek () [0];
                        crtMinMax [1] = crtNode.right.val>
? PathMinMax.peek () [1] crtNode.right.val: pathMinMax.peek () [1];
                        pathNode.push (crtNode.right); pathMinMax.push

                (crtMinMax);} if (crtNode.left! = null) {


                        int [] crtMinMax = new int [2]; crtMinMax [0] =
                        crtNode.left.val <
? PathMinMax.peek () [0] crtNode.left.val: pathMinMax.peek () [0];
                        crtMinMax [1] = crtNode.left.val>
? PathMinMax.peek () [1] crtNode.left.val: pathMinMax.peek () [1];
                        pathNode.push (crtNode.left); pathMinMax.push

        (crtMinMax);} pathMinMax.pop ();} return amplitude;} // 4




public static int getAmplitude3 (TreeNode root) {
        if (root == null)
                return 0;
        return helper2 (root, root.val, root.val);}


private static int helper2 (TreeNode root, int min, int max) {
        if (root == null)
                return max - min; if (root.val
        <min)
                min = root.val; if (root.val>
        max)
                max = root.val;
```

```
                return Math.max (helper2 (root.left, min, max),
helper2 (root.right, min, max));
        }

        public static class TreeNode {
                int val; TreeNode left;
                TreeNode right; TreeNode (int
                x) {

                        val = x;}}



        public static void main (String [] args) {
                TreeNode n1 = new TreeNode (6); TreeNode n2 =
                new TreeNode (4); TreeNode n3 = new TreeNode
                (8); TreeNode n4 = new TreeNode (2); n1.left = n2;
                n1.right = n3; n2. left = n4;




                System.out.println (getAmplitude2 (n1));}}
```